Performance Analysis of the Subgroup Method for Collective I/O

Kwangho Cha, Hyeyoung Cho, and Sungho Kim

Abstract—As many scientific applications require large data processing, the importance of parallel I/O has been increasingly recognized. Collective I/O is one of the considerable features of parallel I/O and enables application programmers to easily handle their large data volume. In this paper we measured and analyzed the performance of original collective I/O and the subgroup method, the way of using collective I/O of MPI effectively. From the experimental results, we found that the subgroup method showed good performance with small data size.

Keywords—Collective I/O, MPI, Parallel Filesystem.

I. INTRODUCTION

BECAUSE many scientific applications such as 'parallel out-of-core' require large data processing, parallel I/O of high performance systems has received many attentions. The research about parallel I/O can be classified into the studies about parallel file system and parallel programming environment. This paper discusses collective I/O in terms of parallel programming environment such as MPI and analyzes the performance of collective I/Os and their variants.

Collective I/O is a technique that handles discontiguous small requests from each computational node[1]. MPI-IO, one of the part of MPI2, also defines collective I/O and MPI implementations such as MPICH[2] and LAM/MPI[3] provide this feature.

The subgroup method for collective I/O is a way of using collective I/O effectively in terms of application programs[4]. The main concept of the subgroup method is that after splitting the entire processes group into several subgroups, only master nodes of each subgroup participate collective I/O. Although the previous work explained the main concept of the subgroup method, it was not enough to verify this suggested scheme. In this paper, we explain the concept of the subgroup method and its overhead in more detail and introduce the performance on the linux cluster system. We could measure the performance of the subgroup method on the PVFS [5], [6] and verify the advantage and the limit of the subgroup method strictly.

Since the subgroup method is an application level approach, its performance is rely on the performance of lower layer such as MPI library, communication protocol, networks and so on. For this reason, our performance analysis is based on the performance measurement on application layer. From the experimental results, we found that, with small data size, especially less than 16KB, the subgroup method showed good performance. We think the experimental result implies

The authors are with the Supercomputing Center, Korea Institute of Science and Technology Information(KISTI), Korea (e-mail: khocha@kisti.re.kr).

it will be useful to use the subgroup method for fine-grained applications.

This paper is organized as follow. Section 2 introduces collective I/O and its related works. Section 3 describes the concept of the subgroup method in detail. The test result of the subgroup method is described in section 4. Finally, we draw the conclusion in section 5.

II. COLLECTIVE I/O

This section describes the concept of collective I/O and the previous works related with collective I/O. When MPI2 was announced, its I/O semantics, MPI-IO, also included the concept of collective I/O. Since MPI is the most prevail parallel programming environment nowadays, in this paper we focus on collective I/O of MPI-IO.

A. Collective I/O

The studies about many parallel applications reported that data from each processes are stored noncontiguously in memory or file system and each process wants to write or read their own data.[1], [7]. Collective I/O handles these requests efficiently by regarding separate and discontiguous I/O requests as a contiguous one. In other words, each process participates an I/O operation for the same file simultaneously. Generally it tries to improve I/O performance by merging separate I/O requests.

There are several implementations of collective I/O such as two-phase I/O and disk direct I/O[7], [8], [9]. In case of two-phase I/O, a data can be accessed efficiently by splitting the access into two phases. In the first phase, processes access data assuming a distribution in memory that results in each process making a single, large, and contiguous access. In the second phase, processes redistribute data among themselves to the desired distribution. The advantage of this approach is it is possible to reduce the file access time by making all file accesses large and contiguous[9].

B. Collective I/O of MPI-IO

MPI-IO defines some types of file operations such as collective, non-collective, blocking, non-blocking, and so on[10], [11]. Collective I/O of MPI-IO means all processes in the same communicator perform I/O operation in the same time and its examples are the functions such as `MPI_File_read_all()`, `MPI_File_write_all()`, and so on.

Before calling a collective I/O function, the filetype should be defined. Because all data from each process should be

.located in a file, each process has their own location in the file. The filetype indicates which portions of file can be accessed by the process and primitive and derived datatype are used to define the filetype. ROMIO[12] is the well known MPI-IO implementations and supports various kinds of file systems such as NFS and PVFS.

Some MPI implementations also provide the performance improvement technique such as access range hints[13], [14]. The access range hint is the access information which is described in application program. By passing the information about the blocks that will be accessed or not to file system, it is possible to manage the prefetch technique efficiently.

III. THE SUBGROUP METHOD FOR COLLECTIVE I/O

In this section, we describe the main concept and the structure of the subgroup method. Additionally the overhead of the subgroup method and its real experimental result on a cluster system is introduced.

A. The Main Concept of the Subgroup Method

When collective I/O is performed, it is required that all processes exchange the information about I/O operations before executing collective write or collective read[7], [8]. In other words, collective I/O is based on collective communications among all processes in the group.

The subgroup method is a style of programming trying to reduce the communication costs by decreasing the number of processes which exchange the I/O information. Like figure 1, in case of the original collective I/O, all processes in the default communicator participate an I/O operation. However in case of the subgroup method, the default communicator is divided into subgroups and only master processes of each subgroup participate collective I/O. They also collect or dispense data of their subgroup which should be written or read. In other words, in case of collective write, master processes gather data from their subgroup member and perform I/O function. For collective read, master processes scatter data to member nodes of their subgroup after I/O operation.

The other effect of the subgroup method is to increase data size. Because data in a subgroup are aggregated, the larger data are passed to collective I/O. Therefore, in case of small data, it can improve the communication-to-computation ratio.

B. The Structure of the Subgroup Method

Figure 2 shows the pseudo-code of the subgroup method. The primitive MPI function for manipulating communicators, `MPI_Comm_split()` is used for generating subgroups and collective communications such as `MPI_Gather()` and `MPI_Scatter()` are used for gathering and distributing data within subgroup respectively.

Like figure 3, two kinds of communicators are newly created and only the processes in the I/O communicator participate in the collective I/O. To create two kinds of the communicator is the first overhead of the subgroup method. We measured the cost of `MPI_Comm_split()` on our linux cluster system and figure 4 shows the result. We can observe that its cost is proportional to the size of the default communicator.
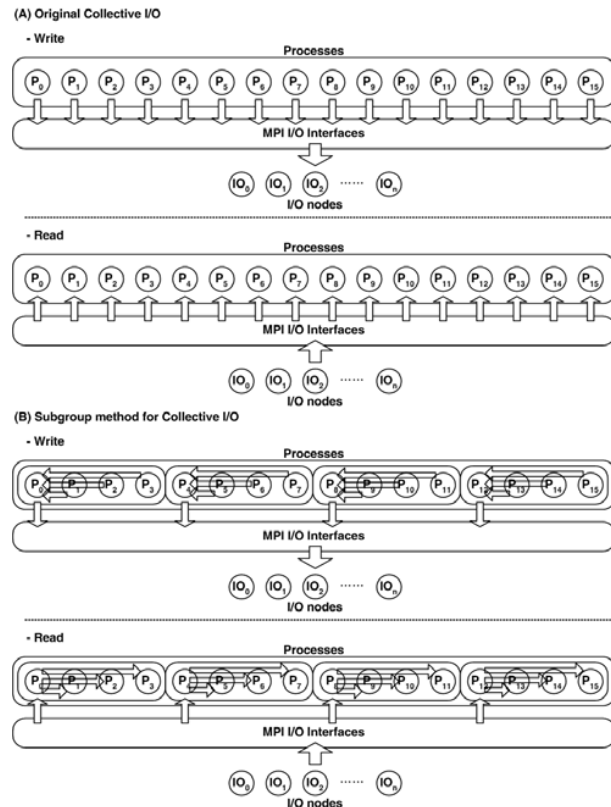

Fig. 1. The concept of original collective I/O and the subgroup method.

```
/* Generate new communicator for subgroups */
MPI_Comm_split(MPI_COMM_WORLD,COLOR,KEY,&NEW_COMM);
MPI_Comm_rank(NEW_COMM, &new_rank);

/* Generate new communicator for the master processes */
if(master_process) set io_color;
MPI_Comm_split(MPI_COMM_WORLD,io_color,SUB_KEY,&IO_COMM);
    ..........
if (root_process) {
/* File Open & Generate filetype */
}
-------------------------------------------------------------
/****** READ ******/       | /****** WRITE ******/
if (master_process) {      | MPI_Gather(..., NEW_COMM);
/* Perform collective  */  | if (master_process) {
/*       read operation */ | /*      Perform collective   */
  MPI_File_set_view(....);  | /*         write operation */
  MPI_File_read_all(....);  |   MPI_File_set_view(....);
}                          |   MPI_File_write_all(....);
MPI_Scatter(..., NEW_COMM); | }
-------------------------------------------------------------
if (root_process) MPI_File_close();
```

Fig. 2. The pseudo-code of the subgroup method.

Figure 5 and 6 show the cost of collective communications, `MPI_Gather()` and `MPI_Scatter()`. As the number of processes and the data size are increased, the performance is also degraded. Because the performance of `MPI_Scatter()` is better than that of `MPI_Gather()`, it is expected that the performance of the subgroup method for collective read is better than that of the subgroup method for collective write on our testbed.
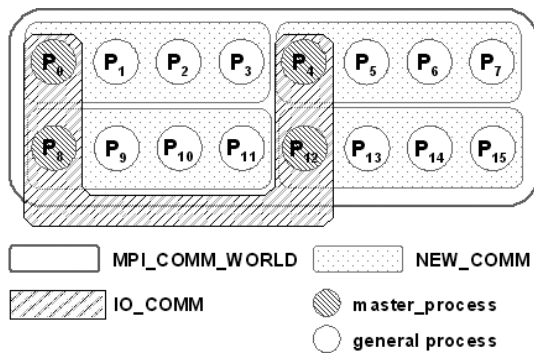
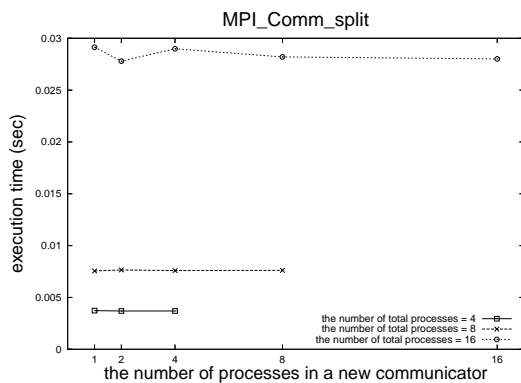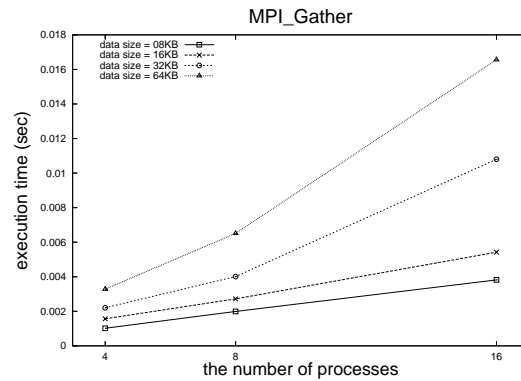Fig. 3.   The new communicators in subgroup method.

Fig. 5.   The cost of MPI_gather().

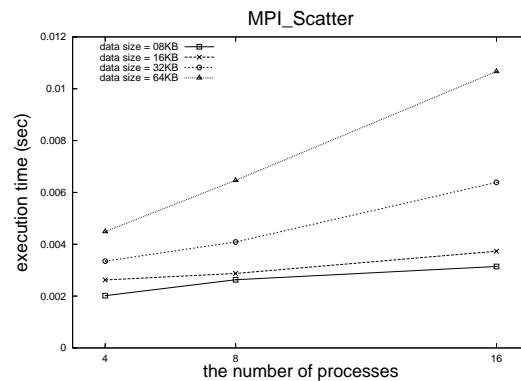Fig. 4.   The cost of MPI_Comm_split().

Fig. 6.   The cost of MPI_Scatter().

IV. Performance Evaluation

To test the performance of the subgroup method, we used 16 nodes cluster system as the testbed like table I. Each node has two AMD processors and all nodes are connected via gigabit ethernet. Because file system should be support MPI-IO, we installed PVFS, one of the famous parallel file systems. Because our testbed is the small scale cluster system, we configured PVFS with 4 I/O servers and 1 meta server. To use MPI-IO semantics properly, we also used MPICH2 library.

Figure 7 shows the result of our experiments. As we expected in the previous section, the performances of the subgroup method for collective read is better than that for collective write. Especially, it shows that when the data size is small, the subgroup method is superior to original collective I/O. When the data size is less than 16KB, the performance improvement for collective write and collective read are upto 53% and 70% respectively. Furthermore, in case of data size is

4KB, the subgroup method shows more improved performance irrespective of the number of processes. This implies that it is possible to consider using of the subgroup method for fine-grained applications.

V. Conclusion

We have studied the efficient collective I/O of MPI and this paper introduces the performance of collective I/O and the subgroup method. Although the subgroup method requires additional MPI functions, it can reduce the number of processes in the communicator for collective I/O. Because it also increases the data size for collective I/O, it can improve its performance.

In our test environment, because the cost of data gathering and scattering in sub communicator were not negligible, the subgroup method with large data failed to improve the performance. On the other hand, in case of using small data, the subgroup method shows good performance. For this reason, it is expected that the subgroup method can be used for fine-grained applications.

This paper is based on the experimental result on the small scale linux cluster system. To obtain more detail characteristics of the subgroup method, we are planning to expand the scale of testbed.

TABLE I
The configuration of testbed

CPU	AMD Opteron 240
CPU/Node	2
The Number of Nodes	16
Memory	1 GB
Network	Gigabit Ethernet
OS	Linux 2.6.9
File System	PVFS 2.6.1
MPI Library	MPICH2-1.0.5

References

[1] John M. May, "Parallel I/O for High Performance Computing," *Morgan Kaufmann*, 2000.
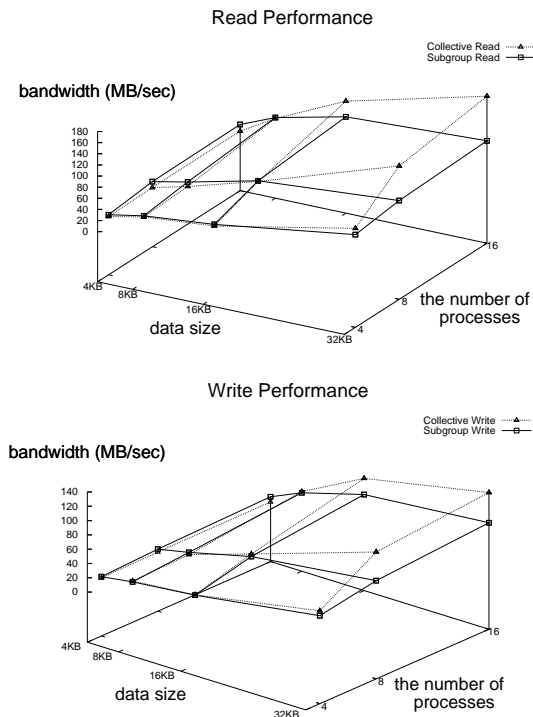
Read Performance

Write Performance

Fig. 7. The read and write performance: For the subgroup method, the size of subgroup was 2 when the number of processes was 4 and in other cases it was set as 4.

on top of GPFS," *Proc. 2001 ACM/IEEE conference on Supercomputing(CDROM)*, pp 17∼17, Nov. 2001.

[2] MPICH-A Portable Implementation of MPI, http://www-unix.mcs.anl.gov/mpi/mpich
[3] LAM/MPI Parallel Computing, http://www.lam-mpi.org
[4] Kwangho Cha, Taeyoung Hong, and Jeongwoo Hong, "The Subgroup Method for Collective I/O," *Proc. The 5th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2004), LNCS 3320*, pp. 301∼304, Dec. 2004.
[5] Avery Ching, Alok Choudhary, Wei-keng Liao, Rob Ross, and William Gropp, "Noncontiguous I/O through PVFS," *Proc. IEEE International Conference on Cluster Computing*, pp 405∼414, 2002.
[6] Philip H. Carns, Walter B. Ligon, III, Robert B. Ross, and Rajeev Thakur, "PVFS: A Parallel File System for Linux Clusters," *Proc. 4th Annual Linux Showcase and Conference*, pp 317∼327, 2000.
[7] David Kotz, "Disk-directed I/O for MIMD multiprocessors," *ACM Transactions on Computer Systems*, Vol. 15, No. 1, pp 41∼74, Feb. 1997.
[8] Rajesh Bordawekar, "Implementation of collective I/O in the Intel Paragon parallel file system: initial experiences," *Proc. 11th international conference on Supercomputing*, pp 20∼27, 1997.
[9] Rajeev Thakur, William Gropp, and Ewing Lusk, "Data sieving and collective I/O in ROMIO," *Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation*, pp 182∼189, 1999.
[10] William Gropp, Ewing Lusk, and Rajeev Thakur, "Using MPI-2: Advanced Features of the Message Passing Interface," *The MIT Press*, 1999.
[11] Hakan Taki and Gil Utard, "MPI-IO on a parallel file system for cluster of workstations," *Proc. 1st IEEE Computer Society International Workshop on Cluster Computing*, pp 150∼157, 1999.
[12] ROMIO: A High-Performance, Portable MPI-IO Implementation, http://www-unix.mcs.anl.gov/romio
[13] Jean-Pierre Prost, Richard Treumann, Robert Blackmore, Carol Hartan, Richard Hedges, Bin Jia, Alice Koniges, and Alison White, "Towards a High-Performance Implementation of MPI-IO on Top of GPFS," *Proc. The 6th International Euro-Par Conference, LNCS 1900*, pp 1253∼1262, Sep. 2000.
[14] Jean-Pierre Prost, Richard Treumann, Richard Hedges, Bin Jia, and Alice Koniges, "MPI-IO/GPFS, an optimized implementation of MPI-IO

Kwangho Cha received the M.E. degree in computer engineering from the Information and Communications University, Daejeon, Korea in 2002. Since 2002, he has been with the Supercomputing Center of Korea Institute of Science and Technology Information, Daejeon, Korea, where he is currently a research staff. He is also the Ph.D. student in computer science at the Korea Advanced Institute of Science and Technology. His research interests include cluster computing, parallel file system, and grid computing. He is a member of the IEEE Computer Society and the Korea Information Science Society.

Hyeyoung Cho received the M.E. degree in computer engineering from the Information and Communications University, Daejeon, Korea in 2004. She is currently a researcher of the Supercomputing Center of Korea Institute of Science and Technology Information. Her interests include cluster system, distributed and parallel file system, and embedded system. She is a member of the Korea Information Science Society.

Sungho Kim received the Ph.D. degree in aerospace engineering from Korea Advanced Institute of Science and Technology, Daejeon, Korea in 1999. He is currently a senior researcher of the Supercomputing Center of Korea Institute of Science and Technology Information, Daejeon, Korea. He performed many national projects related to cluster computer architecture, system software and grid computing technology. He is now one of the key members to design 4th supercomputer of KISTI Supercomputing Center and other related projects. His research interests include cluster computing and embedded computing.